Reusing a Simulation Facility Software Architecture for Embedded Simulation

Frank A. Cioch
Computing and Software Systems
University of Washington, Bothell
Box 358534
18115 Campus Way N.E.
Bothell, WA 98011-8246

(425) 352-5495
cioch@u.washington.edu

Scott Lohrer
Vetronics Technology Center
U.S. Army Tank-Automotive and Armaments Command
AMSTA-TR-R MS 264
Warren, MI 48397-5000

(810) 574-5035
lohrers@tacom.army.mil

Keywords

reuse, legacy systems, evolvability, long-lived systems, technology transfer, software architecture, embedded simulation, distributed simulation, middleware, architectural degradation

Abstract

In 1992 we began the development of a ground vehicle simulation software architecture that was specifically designed for evolvability. This architecture along with its supporting development processes and standards was reused throughout the 1990's on a variety of Army programs. We have now taken this software architecture, which was designed for stationary simulation in crewstations in a laboratory environment and reused it to add embedded simulation capabilities to fielded ground vehicles.

The purpose of embedded simulation is to enhance soldier effectiveness both before and during actual battlefield conditions. As a result, in a fielded ground vehicle, the computer specialist responsible for setting up, starting, controlling, monitoring and stopping the simulation is not present. Unlike in a laboratory environment, in a fielded vehicle the simulation software itself is responsible for performing these functions.

This paper describes how our simulation facility software architecture has been adapted so that it could be reused for embedded simulation. We thus provide a case study illustrating characteristics of a system architecture that allow it to be long-lived by evolving not only to changing requirements due to technological and methodological changes but also changes due to new and previously unthought-of patterns of system utilization. We also share what we have learned about the development of embedded software systems that need to be able to run without human operator intervention.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
23 JUN 2003	**Journal Article**	**22-05-2003 to 22-06-2003**

4. TITLE AND SUBTITLE **Reusing a Simulation Facility Software Architecture for Embedded Simulation**	5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Frank Cioch; Scott Lohrer**	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Computing and Software Systems,University of Washington, Bothell Box 358534,18115 Campus Way N.E.,Bothell,WA,98011**	8. PERFORMING ORGANIZATION REPORT NUMBER **; #13899**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) **U.S. Army TARDEC, 6501 East Eleven Mile Rd, Warren, Mi, 48397-5000**	10. SPONSOR/MONITOR'S ACRONYM(S) **TARDEC** 11. SPONSOR/MONITOR'S REPORT NUMBER(S) **#13899**

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In 1992 we began the development of a ground vehicle simulation software architecture that was specifically designed for evolvability. This architecture along with its supporting development processes and standards was reused throughout the 1990?s on a variety of Army programs. We have now taken this software architecture, which was designed for stationary simulation in crewstations in a laboratory environment and reused it to add embedded simulation capabilities to fielded ground vehicles. The purpose of embedded simulation is to enhance soldier effectiveness both before and during actual battlefield conditions. As a result, in a fielded ground vehicle, the computer specialist responsible for setting up, starting, controlling, monitoring and stopping the simulation is not present. Unlike in a laboratory environment, in a fielded vehicle the simulation software itself is responsible for performing these functions. This paper describes how our simulation facility software architecture has been adapted so that it could be reused for embedded simulation. We thus provide a case study illustrating characteristics of a system architecture that allow it to be long-lived by evolving not only to changing requirements due to technological and methodological changes but also changes due to new and previously unthought-of patterns of system utilization. We also share what we have learned about the development of embedded software systems that need to be able to run without human operator intervention.

15. SUBJECT TERMS
reuse, legacy systems, evolvability, long-lived systems, technology transfer, software architecture, embedded simulation, distributed simulation, middleware, architectural degradation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Public Release**	**23**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Biographical Information

Frank Cioch is a software engineer, with degrees in math, statistics and computer engineering, and a doctorate in Computer and Communications Sciences from the University of Michigan. After obtaining his Ph.D. in 1985, he taught at Oakland University in the greater Detroit area until Autumn 2000, when he started teaching at the University of Washington, Bothell. Dr. Cioch's technical interests derive from his basic interest in software comprehension, both as it relates to software's internal characteristics and to its utilization in a particular environment. His specialty is assessing the degree of fit of software engineering techniques, tools and methods to any given situation, and tailoring their application to enhance their effectiveness. His practical experience includes serving as a contractor for the U.S. Army, consulting for automobile-related companies in the Detroit area and participating in the failure of two start-up companies.

Scott Lohrer is a computer engineer with a degree in electrical engineering. After obtaining his BSEE he worked for a year writing control software for heat treatment furnaces at a private company before coming to work for the Army. Scott has spent over nine years working in the VETRONICS area of TARDEC developing simulation. He enjoys working with the cutting edge technology that is necessary in the Army and seeing it transition to new systems for the soldier.

1. Introduction

In 1992 we began the development of a ground vehicle simulation architecture that was specifically designed for evolvability. The architecture was developed using an evolutionary life cycle model embedded in Boehm's (1988) spiral life cycle model. A discussion of the early versions of the architecture can be found in Cioch, et. al. (1994). The evolvable software architecture was used throughout the 1990's for all simulations developed in the Vetronics Simulation Facility (VSF) at the Tank-Automotive Research, Development and Engineering Center (TARDEC). During this time it was used by a number of teams to support a variety of ARMY programs (Cioch and Sieh, 1999; Cioch, et. al., 2000).

The VSF consisted of crewstations capable of representing different vehicles through both reconfigurable hardware and software, computer generated forces for friendly and threat vehicles, and a stealth vehicle station for monitoring the battle in a computer-generated world. The role of the VSF was to provide asynchronous near-real-time simulations of Army combat ground vehicles that allowed new vehicle concepts to be evaluated by the soldier in a stand-alone mode or in a force-on-force mode. VSF simulations provided realistic mockups of the interiors of combat vehicles, especially their controls and out the window views.

The VSF mission required that TARDEC engineers be capable of quickly developing new vehicle simulations to evaluate new crewstation concepts such as panoramic and helmet mounted displays and new vehicle concepts such as mobility and weapons system models. Therefore, in addition to being used for simulation exercises and demonstrations, the VSF facilities were used by engineers to develop new simulations.

The software architecture that we developed was able to continually evolve into handling more sophisticated simulations as well as technological changes to the VSF itself. Although the VSF is no longer operational, the evolvable software architecture is now being used on embedded simulation, a substantially different type of ARMY program with significantly different requirements. Embedded simulation is an emerging technology in the research and development stage.

The basic idea of embedded simulation is to put simulation capability on fielded ground vehicles. During its operation the embedded simulator would be using actual values from vehicle hardware and software subsystems to perform its functions. The embedded simulation could be run in either operational or stationary mode. In operational mode the mobility and weapons systems of the vehicle are fully operational and the simulator uses values from them to perform its functions. In stationary mode simulated vehicle systems models provide values during the execution of the simulation exercise.

Embedded simulation would be used to enhance soldier effectiveness both before and during actual battlefield conditions. In operational mode the embedded simulator would provide support to the soldier while on the move during battlefield operations by augmenting the soldiers sights with a virtual out the window view of the battlefield terrain. For example, a birds-eye view of the battlefield terrain with the friendly and enemy forces positioned on that terrain could augment the soldier's limited view from the vehicle's sights or Indirect Vision Displays (IVDs). In stationary mode the simulator would provide the soldier with the capability for embedded training and mission rehearsal using a virtual out the window view before the missions were actually carried out.

In the VSF a human operator was always present to set up the simulator, start the required hardware and software systems and monitor the soldier's use of the simulator. In embedded simulation the computer specialist that starts, stops and controls the simulator is not present. On the battlefield, the simulation software itself becomes responsible for self-starting, self-stopping and self-correcting in the event of any malfunction.

The VSF software architecture has been integrated into an embedded simulation software architecture. This new architecture meets new embedded simulation program requirements as well as all of the old VSF evolvability requirements. This paper describes the new architecture and how it addresses the requirements of the embedded simulation program. First we will describe how our system architecture relates to existing knowledge of software and simulation.

1.1 The Problem

In 1992 we started the first version of our object-oriented architecture for our distributed near-real time simulation system [Cioch, Brabbs and Kanter, 1994]. It is basically a message-oriented middleware architecture that is based on the object-oriented design principles of information hiding, collaboration and responsibility, and separation of concerns [Rumbaugh, 1991; Wirfs-Bock, 1990]. Over the years we have been able to evolve the architecture to continue to meet new requirements [Cioch and Sieh, 1999; Cioch, Brabbs and Sieh, 2000].

Over the past decade there has been a growing influence of design patterns in software development [Gamma et. al., 1995; Buschmann, et.al., 1996]. In the past few years, reusable design patterns for distributed system development have appeared [Schmidt, et.al., 2000; http://www.hillside.net] These design patterns are starting to be embodied in commercial off-the-shelf (COTS) components and middleware frameworks for distributed system design. There are now a number of viable candidate frameworks such as CORBA [http://www.omg.org/], ACE/TAO [http://www.cs.wustl.edu/~schmidt] and HLA [https://www.dmso.mil/public/transition/hla/] upon which we could reengineer our architecture.

We are thus faced with the decision of whether to re-engineer our simulation system architecture using one of these middleware frameworks or whether instead to continue to evolve our existing message-oriented middleware architecture, which has been specifically designed to meet our particular requirements. The answer has not been obvious - there are risks involved in adopting a middleware framework. These risks apply not only to distributed simulation design, but in various ways to all tool adoption decisions.

One risk is quality of service (QoS). In order to be a viable alternative, a middleware framework has to provide the same level of performance, security, etc. as our current system architecture. For example, in our case we had have had continuing questions whether CORBA would allow us to meet our distributed system near-real time performance requirements [Gokhale, et.al., 1997; Schmidt, 2003].

A second risk has been the lack of standardization and the possibility of selecting the wrong framework. It is difficult to decide which framework to use – HLA, CORBA, ACE, or another framework. We have been concerned that we would make the wrong choice and would have to re-architect yet again as the industry standard emerged over time. Our decision has been

complicated by the fact that HLA is emerging as a military simulation standard while CORBA is emerging as an industry standard.

A third risk is that the framework would not allow us to meet all key functional requirements, either because it lacked key functionality or because we could not figure out how to use it to meet our needs. The possibility that we could incur the costs of ramping up to learn the new technology, only to find that we could not make it meet our needs, has been a significant concern.

This has been one of the key issues facing both the developers and the users of reusable components. Fischer (1987) argues that to fully exploit the benefits of software reuse, information must be provided that helps software developers determine what the reusable software does, when it is appropriate to use it, how to use it, and how to tailor it for their specific needs. In their survey of reuse issues and research directions, Mili et al. (1995) make the point that successful reuse depends upon locating, evaluating and tailoring the reusable software. They describe two stages for reusing software: acquisition and adaptation. Acquisition requires semantic knowledge for using the component in order to assess the reusable software's usefulness for the problem at hand.

A fourth risk is a possible significant impact of the framework on our existing software development processes. Because the introduction of a new technology typically requires modification of existing software development processes, it can be disruptive and expensive to change to a new technology. Schmidt and Fayad (1997, p.86) describe the lessons they have learned working for many companies over the past decade building and using reusable object-oriented frameworks and applications: "[s] uccessful reuse generally requires the presence of certain key non-technical prerequisites. Many political, economic, organizational, and psychological factors can impede the successful reuse of distributed software." In their discussion of reusing object-oriented frameworks, Sparks et al. (1996) make a similar point - the use of frameworks requires more discipline in design decision making, requiring more technical and management reviews. Jacobson et al. (1997) says that a common misconception about software reuse is that it is primarily a matter of introducing the appropriate technology. It is that and everything else – management, organization, architecture, processes, investment, and persistence.

This risk is not limited to the adoption of reusable components and object-oriented frameworks. In their discussion of the difficulty of transitioning all types of software technology, Fowler and Maher (1992) discuss changes in work habits that a new technology may cause. Gillies and Smith (1994) make the point that the management of people and the effect of change on them is crucial to the successful implementation of software engineering tools and methods. In their discussion of why some organizations are successful while others struggle with CASE implementation, Corbitt and Norman (1989) argue that the introduction of new technology takes place more effectively when the procedure being automated is already an accepted, and valued, procedure. In his discussion of the relationship between CASE planning and the software process, Humphrey (1989) suggests that the most important CASE planning guideline is that an organization develop their process and get it under control before or during CASE installation, but not after. Leonard-Barton (1988) suggests that getting new technologies up and running in

daily operations is at least as challenging as their invention. When discussing ways to evaluate software engineering tools for possible adoption by an organization, Topper (1991) argues that the development team has to adapt the development methods to fit the tool as well as adapt the tool to fit the methods used by the team.

We have found it highly advantageous to develop tailored software engineering processes to support the software architecture [Cioch and Sieh, 2000]. These would need to be modified to support the use of an object-oriented framework such as CORBA, ACE or HLA.

These types of risks may be one of the reasons for the popularity of using object-oriented wrappers to participate in HLA federations [Lohrer and Cioch, 1998]. In September of 1996, the Under Secretary of Defense for Acquisition and Technology issued a mandate that HLA be applicable to all DoD simulations. This mandate caused a problem for agencies with legacy systems that were actively utilized, particularly when funding wasn't available for upgrading the legacy systems. The wrapper approach is to keep your existing architecture to as great an extent as possible and to add an input/output wrapper that takes care of all data exchanges with other simulations in the federation. This allows systems that do not have anHLA software architecture to interact with those that do. An alternative, yet similar approach is to use a commercial Gateway to communicate with other participants within the federation. [Paterson, Hougland, Sanmiguel, 2000] The scheme allows DIS compatible simulations to now participate in an HLA federation. Each scheme has its shortcomings. With the gateway, the user is limited to the federation that the tool will support. With the wrapper, this approach can lead to laborious maintenance to alter FOM data and in each case optimal performance could be questioned.

Given these considerations, we decided to continue to evolve our long-lived legacy system to meet new requirements rather than by re-architecting the system using a collection of interacting components or an object-oriented framework. We have developed a working version, which we have demonstrated for the VETRONICS Technology Testbed program. [Bounker, Brabbs, Adams, 1999].

This paper will illustrate the object-oriented architectural design principles underlying our newly evolved architecture and will show how the architecture relates to the embedded simulation requirements that motivated it. Although we have been satisfied with the new architecture and it meets all of our requirements, we show that our approach has not been risk free. We will see that some architectural degradation has begun to occur as some of the requirements have become significantly different from those that motivated the original system architecture. The case study presented here can thus also be used as an example of the ways in which architectural degradation can occur over time as an object-oriented legacy system evolves.

In the next section of the paper we briefly summarize the VSF software architecture and the evolvability requirements that it had to satisfy. We then discuss in detail the requirements of the embedded simulation program and how the VSF software architecture has been incorporated into the embedded simulation software architecture to meet those requirements.

2. The Purpose of the Evolvable VSF Software Architecture

In 1992 TARDEC initiated the development of the VSF. The VSF was a research and development resource for the evaluation of vehicle concepts. The role of the VSF was to provide asynchronous near-real-time simulations of Army combat ground vehicles emphasizing soldier machine interface development and testing. The VSF was the soldier-in-the-loop portion of a virtual prototyping process, allowing new vehicle concepts to be evaluated by the soldier in a stand-alone mode or in a force-on-force mode.

The VSF consisted of crewstations capable of representing different vehicles through both reconfigurable hardware and software, computer generated forces for friendly and threat vehicles, and a stealth vehicle station for monitoring the battle in a computer-generated world. The VSF components were capable of running stand-alone at TARDEC or interoperating with other defense simulators using the Distributed Interactive Simulation (DIS) interoperability standard.

VSF simulations provided realistic mockups of the interiors of combat vehicles, especially their controls and out the window views. A simulated vehicle was partitioned into UNIX processes that ran in a parallel, distributed fashion on Silicon Graphics computers. The processes transferred their state information to each other through shared memory, message queues and a LAN, allowing them to perform as an integrated vehicle. Each simulated vehicle interacted with friendly and enemy combat systems through a DIS network interface.

The VSF mission required that TARDEC engineers be capable of quickly developing new vehicle simulations to evaluate new crewstation concepts such as panoramic and helmet mounted displays and new vehicle concepts such as mobility and weapons system models. Therefore, in addition to being used for simulation exercises and demonstrations, the VSF facilities were used by engineers to develop new simulations. It was not uncommon for the VSF to support a number of different development programs at the same time, so access to hardware and software was often limited.

In order to achieve its mission, the VSF's simulations and their components had to be evolvable to accommodate both technological and methodological advances. We developed a software architecture that was able to continually evolve into handling more sophisticated simulations. TARDEC engineers used this evolvable software architecture to develop all VSF simulations.

Two teams within TARDEC utilized the software architecture to develop, with contractor support, simulations for a variety of Army programs. One team used the software architecture to develop simulations to support the Bradley Fighting Vehicle and Anti-Armor Advanced Technology Demonstration (A2-ATD) simulation programs. A second team used the software architecture for the Crewman's Associate Future Combat System, Advanced Abrams Crewstation Program and Future Scout Crewstation Prototype.

2.1 The Requirements that the VSF Software Architecture had to Satisfy

The VSF software architecture was derived not only from requirements of the simulations themselves but also requirements concerning the utilization of VSF facilities for simulation development. That is, both the simulations and the VSF facilities had to be evolvable to accommodate new technological advances. Four requirements motivated the development of the evolvable VSF software architecture.

1. Evolvability: Easy to incorporate new vehicle concepts into simulations

Simulations developed suing the software architecture had to be decomposed so that new crewstation and vehicle concepts and models could be readily incorporated. For example, was desirable that a proposed new weapons system model be easily substituted for the existing model so that it could be evaluated (plug-and-play, or at least readily modifiable, software components).

2. Evolvability: Ability to accommodate new hardware and software technology

New software and hardware technologies had to be accommodated by the VSF so it would not become technologically obsolete over time. The facility itself had to be technologically evolvable. As a result, VSF legacy simulations also had to be evolvable to technological changes to the VSF over time. For example, upgrades to more powerful computers and networking technology in the VSF must not result in difficult and costly legacy simulation system upgrades.

3. Interoperability: Simulations comply with evolving military interoperability standards

Simulations developed using the software architecture had to be compliant with evolving military interoperability standards so that they could participate in exercises with other simulations. Originally, this meant compliance with the DIS standard. All DIS compliant legacy simulations had to evolve to become High Level Architecture (HLA) compliant.

4. Reconfigurability: Reconfigurable assignment of simulation processes to VSF computers

VSF simulations had to be able to run on many different combinations of computers in the facility. This was due to not only monetary constraints but also because it was undesirable to have development bottlenecks resulting from over-utilized and under-utilized computers. During development, balanced usage of all computers in the facility was desired. For example, if two teams were working on development, even during peak periods both teams should be able to find free computers in the facility on which to perform development.

2.2 The Evolvable VSF Software Architecture

The VSF software architecture, which has been incorporated into the embedded simulation software architecture, is a scaled-down, application-specific, object-oriented framework for distributed software (Schmidt and Fayad, 1997). It is based on the design principle that vehicle subsystems and their interactions can be used to identify a vehicle simulator's software components and their interactions. A VSF simulation consisted of parallel execution of UNIX processes representing vehicle subsystems, such as mobility and weapons systems. We discovered that providing a minimal set of asynchronous, near-real-time data communication mechanisms could simulate a wide class of vehicle subsystem interactions. This minimal interface included a mechanism for definition of the structure and type of the data to be exchanged and mechanisms for event data and continuous data exchange. The VSF software architecture centers around a Process Interface Unit (PIU) that defines, provides and encapsulates interprocess communication (IPC).

As shown in Figure 1, the PIU binds together independent processes into a cohesive vehicle simulation. It serves as an intermediary for all data exchanged between the software processes. To send data between processes, a sending process sends data to the PIU. The PIU routes the data to the desired destination process. Direct communication between processes does not take place.

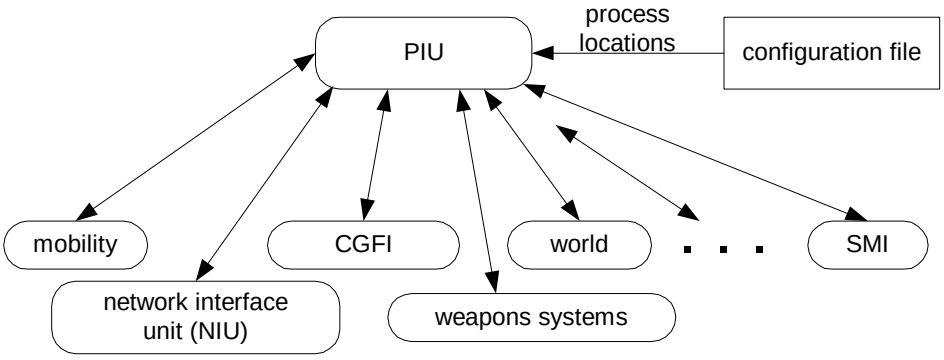

Figure 1: PIU as an Intermediary for Data Communication

Communication between processes is performed using a producer/consumer model. Processes can be both producers and consumers. The VSF architecture is driven by the need to make the routing of data from the producer to the consumer transparent. The producer does not know who the consumers are, whether they are on the same host or how the data is getting to the consumers. The producer and consumer will interface to a set of logical PIU communication functions that will take care of all these details.

As described in detail in Cioch et. al. (2000) and shown in broad terms in Figure 2, the PIU is responsible for both encapsulating IPC and providing an application programmers' interface (API) for VSF developers.

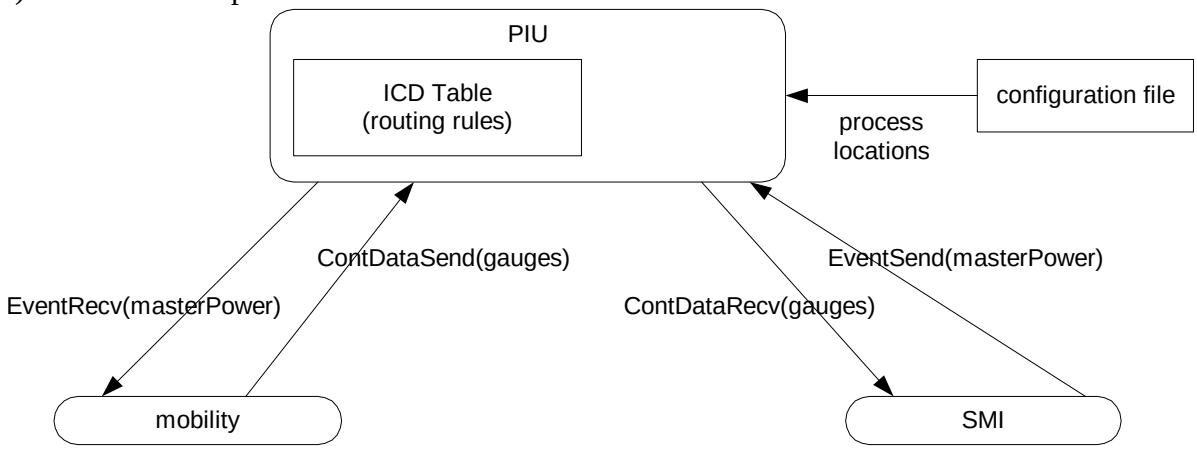

Figure 2: PIU ICD Table and Process Locations File

The PIU encapsulates IPC. The actual mechanisms used to perform data communication, e.g., sockets, shared memory, message queues, is completely hidden within the PIU. The mechanisms used by the PIU to provide data exchange between processes executing on the same host is UNIX System V shared memory and UNIX message queues. The PIU accomplishes data exchange between processes executing on different hosts using sockets connected by FDDI and Ethernet.

As illustrated in Figure 2, a configuration file is used to specify the location of UNIX software processes throughout a LAN. All routing information is invisible to sender and receiver. The actual mechanisms used to perform data communication is completely hidden within the PIU. No

source/destination or process location information appears in any of the send( ) and receive( ) functions executed by the simulation processes.

The PIU provides an API for developers to exchange data between processes. The PIU contains public type definitions of all data to be exchanged along with the definition of a piuComm class containing send( ) and receive( ) member functions for each of these public data types. The piuComm class is linked to each process executable during system build. Each process links in the same piuComm class.

If a developer wants to develop a mobility model that can participate in a VSF simulation, it must be PIU compliant. This means that it must link in the correct version of the PIU, instantiate a piuComm object, and perform the sends and receives expected of it by the other processes. Utilization of the VSF API provided by the PIU has become routine for VSF developers.

In order for the architectural design strategy to work, all interactions between VSF processes have to be fully specified. The VSF interface specification consists of an ICD Table and c++ type definitions for the data to be exchanged. The ICD Table specifies the required data exchange between processes. These process data communication requirements, which describe the interaction of each process with other processes, arise through the identification of the functional requirements of the processes.

A piuComm.h++ header file specifies the actual c++ data types for each data type in the ICD Table. The piuComm class definition contains a pair of send( ) and receive( ) member functions for each data type. A member function EventListSize( ) is provided to allow the client code to see if any events have occurred that need to be handled. A constructor is provided so that a piuComm object can be instantiated for each process involved in the simulation. A list of process IDs is given for all processes involved in the simulation. The argument list of the piuComm class member functions contains no routing information such as consuming processes or computers on which consumers are currently executing.

An example minimal vehicle with soldier-machine interface (SMI) and mobility processes will be used to define a sample ICD Table. SMI sends an event to mobility telling when the vehicle is turned on or off (master power). Mobility sends gauges information containing the heading of the vehicle to SMI for display (gauges osstate). These data communication requirements are captured in the following ICD Table:

DATA TYPES	SMI	mobility
gauges_osstate_type	R	W
master_power_state_type	W	R

Table 1: Minimal Vehicle ICD Table

The following is the public portion of the piuComm.h++ header file for minimal:

```
enum processID_type { SMI, mobility }
struct master_power_state_type { // fields appear here }
struct gauges_osstate_type { // fields appear here }
class piuComm {
public:
```

```
    piuComm(processID_type processID);
    EventSend(master_power_state_type);
    EventRecv(master_power_state_type&);
    int EventListSize();
    ContDataSend(gauges_osstate_type);
    ContDataRecv(gauges_osstate_type&);
private: // implementation details appear here };
```

Mobility and SMI process client code use the resources provided in the piuComm.h++ header file to perform the following tasks. Mobility instantiates a piuComm object and uses it to send its gauges data to SMI. SMI uses its piuComm object to check to see if any events have occurred, and when they do it reads the data into its local copy of gauges.

2.3 How the Architecture Supported the Requirements

1. Evolvability: Easy to incorporate new vehicle concepts into simulations

As program requirements became more solidified and defined, or as a vehicle simulation was used over time, the vehicle modeled changed. If a vehicle needed to incorporate a new feature, such as radio communication, the PIU evolved with the simulation to support the new feature. The team agreed upon changes to the ICD Table and piuComm header file. The team members, including both TARDEC engineers and government contractors located off-site, independently updated their processes from the agreed-upon interface specifications. The PIU developer used the steps given in the *PIU Maintenance Guide* to tailor the PIU for the change. A suite of documents has been developed to support the maintenance process (Cioch, et. al., 1996).

Interface specification was a prerequisite for the software component plug-and-play capability of the VSF architecture. Because all interfaces had been defined in advance over time, enhanced processes could be successively substituted into a simulation as they became available. A new model could be directly substituted for an existing model if the interface was the same. If the new model required modifications to the interface, the developers followed the interface specification process outlined above to make the required modifications.

2. Evolvability: Ability to accommodate new hardware and software technology

The PIU was designed so changes made to the internals of the PIU would not require simulation developers to learn a new programming interface. In particular, the PIU was designed to encapsulate the actual IPC implementation. As new technologies became available, the PIU evolved to make use of new technologies as needed. For example, the PIU was modified to allow IRIX processes running on silicon graphics computers and LINUX processes running on PCs to cooperatively participate in a simulation.

3. Interoperability: Simulations comply with evolving military interoperability standards

The PIU was designed so that it could remain compatible with evolving military interoperability standards. For example, all DIS compliant legacy simulations had to evolve to become High Level Architecture (HLA) compliant. The way in which the PIU architecture addressed this technology transition process is described in Lohrer and Cioch (1998).

4. Reconfigurability: Reconfigurable assignment of simulation processes to VSF computers

VSF simulations were easily configured to run on many different combinations of Silicon Graphics computers in the facility. The PIU provided IPC while the simulation was running in

degraded mode. For example, when enough machines were not available, the PIU was configured through the process locations configuration file to run on the available machines. As expected, the simulator performance was degraded, but the simulation still ran. The ability to run in degraded mode was desirable during development work. The ability to run in degraded mode allowed for work to be done when all required resources were not available. Furthermore, because process locations were contained in the configuration file and were easily changed, many developers could work in the facility at the same time. This allowed for high utilization of facility resources.

3. Embedded Simulation

The basic idea of embedded simulation is to put simulation capability on a fielded ground vehicle. Simulation could be used to enhance soldier effectiveness both before and during actual battlefield conditions. The vehicle could be run in two ways. It could be in full operational mode with the simulator providing support to the soldier while on the move. In this case the soldier would be using the real out the window view or a combination of real and virtual out the window views. Second, the vehicle could be in stationary mode and the simulator would provide the soldier with the capability for embedded training or mission rehearsal using a virtual out the window view.

There are a significant number of technical obstacles that must be overcome before embedded simulation can be used to enhance soldier effectiveness. The purpose of embedded simulation research and development work is to study the technological feasibility of embedded simulation. If successful, embedded simulation capability could become part of the standard package of hardware and software capabilities of all ground vehicles fielded by the U.S. Army.

The Simulation, Training and Instrumentation Command (STRICOM) and TARDEC are working together in this effort (Bounker, et. al., 1999). The embedded simulation program fits under the general Army umbrella of horizontal technology integration (HTI), which strives to reduce overall development costs through the development of configuration items that can be used on different platforms.

At TARDEC, the embedded simulation effort is part of the Vehicle Electronics (VETRONICS) Technology Testbed (VTT), supported by the Inter-Vehicle Electronics Suite (IVES) Science Technology Objective (STO). The VTT program objective is to advance and apply the technologies required for reduced crew operations of close combat vehicles. Embedded simulation is an integral part of this effort.

This paper describes the embedded simulation software architecture and the requirements that motivated its design. Although we have moved from stationary simulation in a simulation facility to embedded simulation in a fielded ground vehicle, this architecture incorporates and reuses the architecture of the VSF described above.

A prototype of our embedded simulation system was developed for the Interservice/Industry Training, Simulation, Education Conference in 1999. That system was then enhanced and a number of experiments have been conducted using the software architecture described in this paper. The embedded simulation system has been demonstrated as part of a futuristic scout vehicle that would be a part of the Army's Future Combat System (FCS). These demonstrations were a result of a consortium of Army Research Development and Engineering Centers (RDEC) that make up the RDEC Federation. These demonstrations occurred at the 2000 and 2001

Simulation and Modeling Acquisition Requirements Training (SMART) conferences, where the vehicle's operational embedded simulation enhancements were demonstrated. At the Armor Conference 2001 at Fort Knox, Kentucky, the embedded training features of the embedded simulation system were demonstrated.

The embedded simulation program has a number of different types of requirements that need to be satisfied by the overall simulation software architecture. We will next describe these requirements and then move on to the software architecture that has been developed to support them.

3.1 Embedded Simulation Program Requirements

In this section the key requirements of the embedded simulation program are described. This discussion will be used to set a context for the requirements of the embedded simulation software architecture that is the focus of this paper.

1. Enhance the vehicle with simulation capability

The vehicle must be enhanced with simulation capabilities that promise to make the soldier more effective on the battlefield. Three different types of simulation support need to be provided to the soldier: support for battlefield operations, embedded training and mission rehearsal.

Simulation capabilities must support battlefield operations. While in operational mode the weapons and mobility systems of the vehicle are operational and simulation is providing enhanced capabilities to the soldier during actual battlefield conditions. The out-the-window views are mixed: some are real and some are virtual views generated by the simulator. An example battlefield operations capability is battlefield visualization, which gives the soldier a simulator-generated virtual birds-eye view of the terrain on which the battle is taking place, with the virtual placement of both friendly and enemy forces

Simulation capabilities must include embedded training. While in training mode the weapons and mobility systems of the vehicle are inoperable and the out-the-window views are virtual views generated by the simulator. An example embedded training capability is sustainment training, which gives the soldier the opportunity to conduct training scenarios and see how current performance compares to the results of previous training sessions. The soldier is able to make comparisons by utilizing the After Action Review and Playback after the training sessions.

Simulation capabilities must include mission rehearsal. While in mission rehearsal mode the weapons and mobility systems of the vehicle are inoperable and the out-the-window views are virtual views generated by the simulator. Mission rehearsal provides the capability for a soldier to walk through a battle plan in the virtual world generated by the simulator before actually doing it under actual battlefield conditions.

2. Support a family of vehicles

The system that is developed must be developed as a separate independent system that can be installed in fielded ARMY combat vehicles.

3. Automate simulator operator capability

For simulation exercises run in the VSF there was a simulation operator, a computer specialist that assisted the soldier in setting up and starting the simulation, monitoring the simulation exercise, and stopping the simulator when the exercise was over. In a fielded vehicle there will

only be the soldier and soldiers are not expected to be computer specialists. Thus, the simulation operator functionality must be automated to as great an extent as possible.

There are five different types of simulator operator capabilities that need to automated:

1. Setting up configuration files so that they have the desired values.
2. Starting up the computers and the simulation software processes.
3. Starting up other software systems that need to interact with the simulator.
4. Monitoring and controlling the system during execution and handling error conditions.
5. Shutting down the system and performing necessary housekeeping tasks.

4. Synchronicity

Ground vehicles operate synchronously. The embedded simulator must be synchronized so that it interacts in a lock-step way with the vehicle itself. For example, readings of vehicle mobility and weapons systems characteristics provided to the simulator must be current and accurate. Second, the virtual and real worlds must be in alignment. For example, the location of the vehicle, as well as friendly and enemy forces, in the virtual world generated by the simulator must be exactly the same as the vehicle's actual location in the real world. Without these capabilities the simulator would not be able to support battlefield operations.

5. Ruggedization

The simulator hardware and software must fit on the vehicle and be capable of running off of the vehicle's power supply. Thus, the system must be ruggedized per military standards for an onboard computer system.

6. Depot-mode operation

Within DoD, Embedded Simulation (ES) for training and Simulation Based Acquisition is a new and untested concept. The purpose of the ES program is to study its technological feasibility before a full commitment to it is made. The prototype must be developed so that this technology transfer can be accomplished in stages rather than all at once.

Thus, the system, when in Training State, currently requires too much power to be able to operate when the vehicle is in the field. In order to execute the embedded training feature, the vehicle has to be in a "depot" mode where it has access to either a generator or wall outlet where the vehicle can receive the additional power through its NATO Slave access. The reason for this is that embedded training requires that all the Indirect Vision Displays (IVD) and the Sights be simulated. This requires the resources of an additional six computers to render the driver's IVDs. Consequently, the system must be reconfigurable and be able to support different combinations of computers and simulation process allocation to computers.

7. Each of the four VSF requirements

Each of the four VSF requirements is still a requirement of the embedded simulation program:

1. In order to support the embedded simulation program, the system must to be able to be easily and readily upgraded to incorporate new ideas in embedded training, battlefield visualization, and battlefield operational support.
2. In order to be compatible with other military systems, the system must be capable of interoperating with military interoperability standards such as High-Level Architecture (HLA).

3. In order to be long-lived and evolvable, the system must be able to be modified to incorporate new hardware and software technology as it becomes available.
4. In order to support its depot-mode requirement, the system must be reconfigurable.

3.2 Embedded Simulation Software Architecture

The embedded simulation software architecture is shown in Figure 3. Bold solid lines represent communication paths related to the PIU-based VSF architecture and dotted lines represent new lines of communication that will now be described.

The software architecture follows the HTI A-Kit/B-Kit paradigm. The vehicle hardware and software interact directly with a system called the A-Kit. The A-Kit is vehicle specific and contains the necessary hardware and software to connect to the actual vehicle hardware and software. As part of the HTI paradigm, an A-Kit is developed for every vehicle. The description of the A-Kit is not within the scope of this paper.

The simulation hardware and software is contained in an independent system called the B-Kit. The B-Kit starts automatically upon vehicle startup and shuts down automatically upon vehicle shut down. The soldier interacts with the simulation while performing embedded training and mission rehearsal via a touch screen, programmable push buttons and a joystick that are part of the standard vehicle hardware. No new additional hardware is required. Vehicle and soldier data resulting from normal system operation like vehicle location and travel speed and direction are sent directly from the vehicle's A-Kit to the B-Kit. The B-Kit contains an A-Kit Interface process that is responsible for all system interaction with the vehicle through the vehicle's A-Kit.

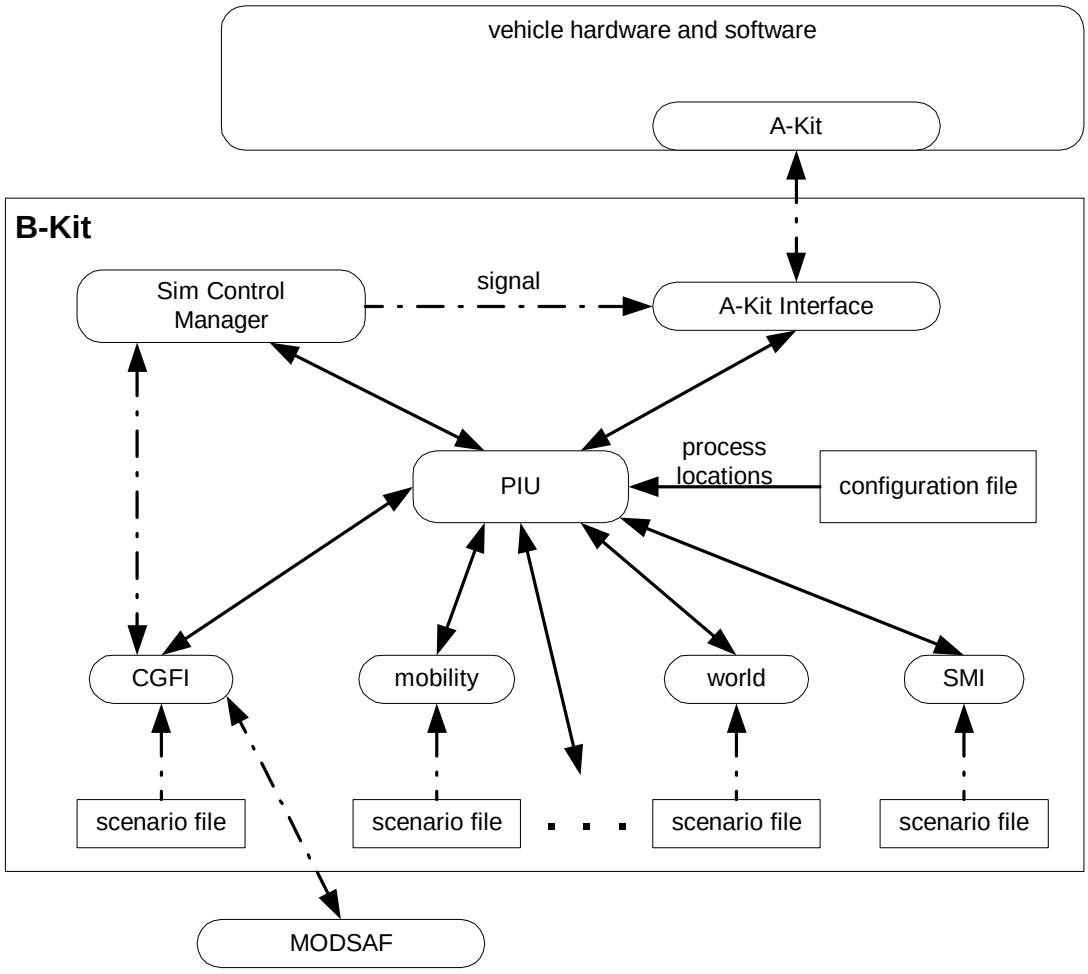

Figure 3. Embedded Simulation Software Architecture

The Simulation Control Manager is the portion of the B-Kit responsible for automating the simulation operator capabilities and keeping the simulation synchronized with the vehicle. In order to control the simulation it communicates with selected simulation processes, currently A-Kit Interface and CGFI, using dedicated specialized methods in addition to communicating with them via the PIU.

The PIU-based VSF architecture is an integral part of the B-Kit architecture. As in the VSF, the PIU is responsible for interprocess communication. The A-Kit Interface uses the PIU to send vehicle data to and from the simulation processes so that the simulator has data from which to function. The A-Kit Interface uses the PIU to send and receive A-Kit simulation control data to and from the Simulation Control Manager so that the latter can control the simulation.

3.2.1 Synchronicity, start-up and shut down - State Machine Behavior

The embedded simulation software architecture has been designed as a state machine to ensure that the simulation and the vehicle are synchronized. This was a substantial shift from the VSF architecture, where all processes operated asynchronously. In this section it will be shown that at the architectural design level, this was achieved by having all simulation processes utilize the ICD Table to send vehicle state data. At the detailed design level, the simulation control manager

was designed to serve as the state controller. At the implementation level, all processes utilized the vehicle state data to remain synchronized.

A ground vehicle equipped with embedded simulation capability can be in one of five states: power up, operational, training, mission rehearsal and power down. The vehicle's s A-Kit and the B-Kit's A-Kit interface send heartbeat data back and forth. This heartbeat is used for two purposes: to tell the other whether it is still up and running and if so, what state it is currently in. The A-Kit Interface uses the PIU to inform the Simulation Control Manager about changes to the vehicle state.

When a vehicle state change occurs, the Simulation Control Manager uses the PIU to tell all of the simulation processes to change state. Most state changes require that new simulation processes and additional computers be started and some of the currently running processes and/or computers be stopped.

The primary reason why software and hardware starting and stopping must be performed during state changes is that if they were not, then the simulator would not be able to run off of the vehicle power supply. It would require too many computers and too many hardware devices be run simultaneously to accommodate all five vehicle states. The required amount of vehicle hardware and software that need to be running at any given time varies depending upon vehicle state:

1. Depending upon the vehicle state, vehicle IVD's can switch between a real out the window view to a virtual out the window view. When a real view is all that is required, image generation software need not be running for that IVD or sight.
2. Depending upon vehicle state, additional computers and supporting software systems need to be started. For example, in training state a separate computer is required to run the drivers IVD's for mission rehearsal and embedded training. It is undesirable to run these additional computers during vehicle operational state.
3. Depending upon vehicle state, the behaviors of the Simulation Processes themselves change. For example, in training and mission rehearsal state the processes must record all behaviors for possible playback to the soldier. This additional functionality is neither required nor desired during vehicle operational state.

As a result of a vehicle state change, the Process Locations file of the PIU changes and the ICD Table data passed between processes that are running also changes. Therefore, all of the processes participating in IPC communication via the PIU have to be stopped and possibly re-started to reflect the new vehicle state. All of this is performed under the control of the Simulation Control Manager.

The Simulation Control Manager sends a state change message to all simulation processes running the PIU. It then shuts them down and restarts them. All Simulation Processes send a ready message to the Simulation Control Manager after they are restarted and ready for data transfer via the PIU. In order to avoid race conditions, a signal is used between the A-Kit Interface and the Simulation Control Manager to ensure that they are both communicating with the newly loaded PIU rather than the previously loaded PIU. A separate line of communication is also required between the Simulation control Manager and the Computer Generated Forces Interface Process because when necessary the latter is responsible for starting an additional process and communicating with the MODSAF system that is running on that computer.

At the architectural design level, to achieve this synchronous behavior we changed the way that we utilized our ICD Table. In the VSF, all ICD Table data was simulation-related data. In the embedded simulation architecture, we added vehicle state data to the ICD Table. By sending vehicle state data to each other via the PIU, each process can remain synchronized with the vehicle state. The enhanced ICD Table for the minimal example described above is given in Table 2. All processes read and write the vehicle state data.

DATA TYPES	SMI	mobility	Simulation Control Manager	A-Kit Interface
gauges_osstate_type	R	W		
master_power_state_type	W	R		
vehicle_state_type	R/W	R/W	R/W	R/W

Table 2: Minimal Embedded Simulation Vehicle ICD Table

Furthermore, in the VSF the ICD Table was tailored to a specific simulation. In the embedded simulation architecture, each vehicle state has its own IPC requirements that can be captured in a different ICD Table. This is because in each state different simulation processes are running and each is performing some state-related functions that result in different ICD Table data transfer. Instead of maintaining separate ICD Tables, the union of all ICD Tables is constructed to account for all possible process sends and receives in all vehicle states.

For example, in the VSF mobility would always be sending gauges information to SMI. However, even in the minimal embedded simulation ICD Table, the mobility process will not be sending gauges information to SMI during operational state, only during training and mission rehearsal. All possible reads and writes during all possible vehicle states are captured in the embedded simulation ICD Table. This is a similar idea to the definition of a shared HLA FOM that defines the entirety of possible federate communication in a federation (HLA web page reference).

This discussion shows how the synchronicity, start-up and shutdown simulator operator requirements are met. The next section describes how the configuration file set-up and monitoring and control requirements were met.

3.2.2 Simulation operator set-up, and control – Scenario Files

During training and mission rehearsal, events must be generated at various points in time or when the vehicle reaches pre-specified locations. In the VSF, a simulation operator would be there to generate the data at the specified trigger points. In the embedded simulation program this operator is not present.

To solve this problem, scenario files were constructed that contained two types of data: simulation scenario data and simulation control data. The simulation control data contains both the data to be sent and a specification of the trigger point. A standard format was developed so that all possible types of trigger points could be specified. Table 3 shows an example scenario file for the Command and Control (C2) process.

The first section of the configuration file is meant for initialization data, in this case the C2_Enable flag and the Time variable for sending Position reports. The remainder of the file describes the specific scenario behavior for the process.

C2_ENABLE	1	
TIME	300	
RECV	CALL_FOR_FIRE	ROGER_OVER

Table 3: Example Scenario File

With an operator, the operator was required to monitor three types of event conditions: time, event and position. For time based events, the operator would keep an eye on the amount of simulation time that had expired and then at the specified moment, execute the desired event. This is accomplished one of two ways with the scenario file. For events that are to occur at some set frequency, such as the Position Reports in our example, a time variable is set to the desired number of seconds. For events that are to be triggered at a particular instance in the simulation time, an alternate approach would be used. In this case the trigger would be the number of seconds that have passed in the simulation. The process responsible for causing the event would have to poll the Simulation Control Manager for the current simulation time. When the desired time has occurred then that event would be triggered. This method wasn't implemented because the need didn't exist; however, the capability is there.

Another condition that an operator would normally have to respond to would be some type of an event, such as an incoming radio message. From the above table the event trigger in this case would be a RECV message of type CALL_FOR_FIRE. When this message is received, the operator would respond with a voice or text acknowledgement. Utilizing the scenario file, the trigger condition is defined as RECV, the constraint is of type CALL_FOR_FIRE and the response is a string containing "Roger Over". This string is read into the process and then sent out to the rest of the simulation.

The last interaction that an operator would perform would be to start an event when the simulated entity crossed a specified plane within the terrain. A scenario may call for an enemy to change positions when the simulated entity reached a particular destination. The operator would then have to execute some command through the CGF GUI to relocate the enemy vehicles. In the ES environment, the scenario file would contain the terrain location as the trigger condition, a tolerance and then the action that is to take place. The action that is to occur would be commands to the CGF that would be routed through the CGFI process using the above example.

This discussion illustrates how utilizing the scenario files, the operator set-up and control requirements were met.

3.3 Relationship Between Program Requirements and Software Architecture

In this section the embedded system program requirements are related to the software architecture.

1. Enhance the vehicle with simulation capabilities

The specific embedded training, mission rehearsal and battlefield operation support capabilities of the simulator are the responsibility of the Simulation Processes. Each of these processes is the responsibility of a software developer.

2. Support a family of vehicles

This requirement is supported by the HTI A-Kit/B-Kit paradigm. The portion of the B-Kit architecture that deals with this requirement is the A-Kit Interface module. By separating into a single module the responsibility for interacting with the vehicle's A-Kit, the impact of change due to differing vehicle A-Kits is localized to one module.

3. Automate simulator operator capability

As described above, this requirement is satisfied primarily by the cooperation between the A-Kit Interface, the Simulation Control Manager, the PIU, and the Simulation processes, under the direction and control of the Simulation Control Manager.

4. Synchronicity

The vehicle runs synchronously and the PIU runs asynchronously. The B-Kit architecture described above reconciles these different modes of operation. This requirement is satisfied by the cooperation between the A-Kit Interface, the Simulation Control Manager, the PIU, and the Simulation processes, under the direction and control of the Simulation Control Manager.

5. Ruggedization

The transition needed to be made from IRIX to LINUX and from silicon graphics mainframes to PCs in ruggedized boxes. This requirement is satisfied by the PIU and was one of the primary reasons that the PIU was retained as a key portion of the architecture rather than moving to HLA or CORBA. The way in which the PIU-based architecture satisfies this requirement is described in detail in Cioch and Sieh (1999) and is summarized above.

6. Depot-mode operation

This requirement is satisfied by the PIU and was one of the primary reasons that the PIU was retained as a key portion of the architecture rather than moving to HLA or CORBA. The way in which the PIU-based architecture satisfies these requirements is described in detail in Cioch and Sieh (1999) and is summarized above.

7. Each of the four VSF requirements

Because the PIU is incorporated into the B-Kit architecture, the B-Kit architecture can satisfy each of the four existing VSF requirements. The way in which the PIU-based architecture satisfies these requirements is described in detail in Cioch and Sieh (1999) and is summarized above.

4. Conclusions

In this paper we presented the current state of our continuing case study involving the evolution of a long-lived object-oriented simulation system. We have demonstrated that through the careful application of object-oriented design principles and associated development processes, it

is possible to continue to update, with minimal architectural degradation, an object-oriented legacy simulation system software architecture to meet new requirements, even when the requirements change significantly.

In general, two significant lessons learned over the course of our decade of research are that (1) the object-oriented design principles of information hiding and separation of concerns can be an effective way to promote software architecture reuse, and (2) it is advantageous to develop tailored software engineering processes to accompany the architecture.

In this paper we have focused on point (1) by describing how we have used the object-oriented design principles in our software architecture and how they relate to our evolvability requirements for embedded simulation. We have demonstrated that even though our requirements changed significantly when we moved to an embedded simulation environment, we were able to evolve our system architecture to meet those new requirements.

As one might expect, over time our requirements are becoming increasingly dissimilar from the original requirements. As illustrated in the earlier description of the architecture, we have had to sidestep the PIU in some cases, indicating that we are starting to experience some architectural degradation in our system. It is possible that the next iteration of the architecture will require a more significant architectural reengineering and the utilization of components or a middleware framework.

The risks of adopting a middleware framework are also becoming smaller. As of this writing, an ACE/TAO real-time CORBA ORB is available for real-time applications [Schmidt and Buschmann, 2003; http://www.cs.wustl.edu/~schmidt] so the QoS risk in using a middleware framework appears to be getting smaller. With the increased literature on design patterns that underlie the middleware frameworks, the risk of our being able to effectively utilize a framework based on these patterns is also becoming smaller. Through the utilization of new software development technologies such as aspect-oriented programming [http://www.aspectj.org; http://aosd.net/], middleware frameworks are becoming increasingly versatile and widely applicable. At some point the economic factors may swing the balance to our using components or a middleware framework to meet our future requirements.

5. Acknowledgments

This work was supported in part by TARDEC under Contract No. DAAE07-94-C-R006 and under the auspices of the U.S. Army Research Office Scientific Services Program Contract No. DAAH04-96-C-0086. We would like to thank all of the VSF team members for their contributions to the VSF and embedded simulation software architectures.

6. References

Boehm, B., "A Spiral Model of Software Development and Enhancement," *Computer*, Vol. 21, No. 5, May 1988, pp. 61-72.

Cioch, F. A., J. Brabbs and S. Kanter, "Using the Spiral Model to Assess, Select and Integrate Software Development Tools," *Proceedings of the 3rd Symposium on Assessment of Quality Software Development Tools*, IEEE, 1994, pp. 14-28.

Cioch, F.A. and L. Sieh, "A Software Architecture for Interoperable, Evolvable, Near Real-Time Simulations," *Simulation*, Vol. 72 No. 2, 1999, pp. 78-89.

Cioch, F.A., J. Brabbs and L. Sieh, "The Impact of Software Architecture Reuse on Development Processes and Standards," *Journal of Systems and Software*, Vol. 50, Issue 3, March 2000, pp. 221-236.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.

Wirfs-Brock, R., B. Wilkerson and L. Wiener, *Designing Object-Oriented Software*, Prentice-Hall, 1990.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, Vol. 1: A System of Patterns*, Wiley, 1996.

Schmidt, D.C., M. Stal, H. Rohnert and F. Buschmann, *Pattern-Oriented Software Architecture, Vol. 2: Patterns for Concurrent and Networked Objects*, Wiley, 2000.

References to patterns can be found at http://www.hillside.net

References to CORBA can be found at http://www.omg.org

References to ACE/TAO can be found at http://www.cs.wustl.edu/~schmidt

References to the HLA Standard can be found at at https://www.dmso.mil/public/transition/hla/

Gokhale, A., Schmidt, D. C., Harrison, T., and G. Parulkar, "Towards real-time CORBA", *IEEE Communications Magazine*, Vol. 14, No. 2, February 1997.

Schmidt, D. and F. Buschmann, "Patterns, Frameworks, and Middleware: Their Synergistic Relationship," *Proceedings of the 25th International Conference on Software Engineering*, May 2003, IEEE, pp. 694-704.

Fischer, G., "Cognitive View of Reuse and Redesign," *IEEE Software*, Vol. 11, No. 5, July, 1987, pp. 60-72

Mili, H., Mili, F. and A. Mili, "Reusing Software: Issues and Research Directions," *IEEE Transactions on Software Engineering*, Vol. 21, No. 6, June 1995, pp. 528-561.

Schmidt, D. C. and M. E. Fayad, "Lessons Learned Building Reusable OO Frameworks for Distributed Software", *Communications of the ACM*, Vol. 40, No. 10, October 1997, pp. 85-87.

Sparks, S., Benner, K. and C. Faris, "Managing Object-Oriented Framework Reuse,", *Computer*, Vol. 29, No. 9, September 1996, pp. 52-61.

Jacobson, I,, Griss, N. G. and P. Jonsson, "Making the Reuse Business Work," *Computer*, Vol. 30, No. 10, October 1997, pp. 36-42.

Fowler, P. J. and J. H. Maher, Jr., "Foundations for Systematic Software Technology Transition," *1992 SEI Technical Review*, Software Engineering Institute, Carnegie-Mellon University, January 1992.

Gillies, A. C. and P. Smith, *Managing Software Engineering: CASE studies and solutions*, Chapter 7, "People Matter," Chapman & Hall, London, 1994.

Corbitt, G. F., and R. Norman, "Implementation: The Operational Feasibility Perspective," *Journal of Systems Management*, Vol. 40, No. 5, May 1989, pp. 32-33.

Humphrey, W. S., "CASE Planning and the Software Process," Software Engineering Institute Technical Report, CMU/SEI-89-TR-26, ESD-TR-89-34, May 1989.

Leonard-Barton, D. "Implementation as Mutual Adaptation of Technology and Organization," *Research Policy*, Vol. 17, No. 5, October 1988, pp. 251-267.

Topper, A., "Evaluating CASE Tools: Guidelines for Comparison," *American Programmer*, Vol. 4, No. 7, July 1991, pp. 12-20.

Daniel J. Paterson, Erik S. Hougland, Ph.D., Juan J. Sanmiguel; "A Gateway/Middleware HLA implementation and the extra Services that can be provided to the Simulation", 2000.

Schmidt, D. C. and M. E. Fayad, "Lessons Learned Building Reusable OO Frameworks for Distributed Software", *Communications of the ACM*, Vol. 40, No. 10, October 1997, pp. 85-87.

Cioch, F. A., M. Palazzolo and S. Lohrer, "A Documentation Suite for Maintenance Programmers," *Proceedings of the International Conference on Software Maintenance*, IEEE, 1996, pp. 286-295.

Lohrer, S. and F. A. Cioch, "Using Layered API's to Reduce HLA Transition Costs", *1998 Spring Simulation Interoperability Workshop Proceedings* (CD-ROM), Simulation Interoperability Standards Organization, Orlando, FL, March 1998.

Bounker, P., J. Brabbs and C. Adams, "Low Cost Embedded Simulation System for Ground Vehicles," *Proceedings of the 1999 Interservice/Industry Training, Simulation and Education Conference* (CD-ROM), National Training Systems Association, Orlando, FL, November 1999.

References to Real-Time CORBA can be found at http://www.cs.wustl.edu/~schmidt

References to aspect-oriented programming can be found at http://www.aspectj.org and http://aosd.net/